

```
BBBBBBBBBBBBBB      AAAAAAAAAA      SSSSSSSSSSSSSS      RRRRRRRRRRRR      TTTTTTTTTTTTTTTT      LLL
BBBBBBBBBBBBBB      AAAAAAAAAA      SSSSSSSSSSSSSS      RRRRRRRRRRRR      TTTTTTTTTTTTTTTT      LLL
BBBBBBBBBBBBBB      AAAAAAAAAA      SSSSSSSSSSSSSS      RRRRRRRRRRRR      TTTTTTTTTTTTTTTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAAAAAAAAAAAAAAAAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAAAAAAAAAAAAAAAAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAAAAAAAAAAAAAAAAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSSSSSSSSSSSSS      RRR      RRR      TTT      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSSSSSSSSSSSSS      RRR      RRR      TTT      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSSSSSSSSSSSSS      RRR      RRR      TTT      TTT      LLL
LLLLLLLLLLLLLLLLLLLL
```

BBBBBBBB	AAAAAA	SSSSSSSS	VV	VV	AAAAAA	LL
BBBBBBBB	AAAAAA	SSSSSSSS	VV	VV	AAAAAA	LL
BB	AA	SS	VV	VV	AA	LL
BB	AA	SS	VV	VV	AA	LL
BB	AA	SS	VV	VV	AA	LL
BB	AA	SS	VV	VV	AA	LL
BBBBBBBB	AA	SSSSSS	VV	VV	AA	LL
BBBBBBBB	AA	SSSSSS	VV	VV	AA	LL
BB	AAAAAAAA	SS	VV	VV	AAAAAAAA	LL
BB	AAAAAAAA	SS	VV	VV	AAAAAAAA	LL
BB	AA	SS	VV	VV	AA	LL
BB	AA	SS	VV	VV	AA	LL
BBBBBBBB	AA	SSSSSSSS	VV	VV	AA	LLLLLLLLLL
BBBBBBBB	AA	SSSSSSSS	VV	VV	AA	LLLLLLLLLL

```

LL          IIIII
LL          IIIII
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LLLLLLLLLLL IIIII
LLLLLLLLLLL IIIII
SSSSSSSSS
SSSSSSSSS
SS
SS
SS
SS
SSSSSS
SSSSSS
SS
SS
SS
SS
SSSSSSSSS
SSSSSSSSS

```

(2) 72
(3) 193
(15) 812
(16) 840
(17) 901
(20) 1057

DECLARATIONS

BASSVAL_x - convert text to floating
RGET - get next character
MUL10_R9 - multiply FAC by 10 and add digit in R3
BASSVAL_L - ; convert text (integer) to longword
BASSVAL_P - convert text to packed decimal


```
0000 1 .TITLE BASSVAL ; Convert text to numeric
0000 2 .IDENT /2-004/ ; File: BASVAL.MAR Edit: MDL2004
0000 3
0000 4 *****
0000 5 *
0000 6 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY *
0000 7 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS. *
0000 8 * ALL RIGHTS RESERVED. *
0000 9 *
0000 10 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
0000 11 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *
0000 12 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *
0000 13 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
0000 14 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *
0000 15 * TRANSFERRED. *
0000 16 *
0000 17 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *
0000 18 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *
0000 19 * CORPORATION. *
0000 20 *
0000 21 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *
0000 22 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL. *
0000 23 *
0000 24 *
0000 25 *****
0000 26
0000 27
0000 28 FACILITY: RTL BASIC language support
0000 29 ++
0000 30 ABSTRACT:
0000 31
0000 32 Performs conversion of character strings containing numbers to
0000 33 floating datatypes.
0000 34
0000 35 --
0000 36
0000 37 VERSION: 2
0000 38
0000 39 HISTORY:
0000 40
0000 41 AUTHOR: R. Will, CREATION DATE: 1-Mar-79
0000 42
0000 43 MODIFIED BY:
0000 44
0000 45 R. Will, 1-Mar-79: VERSION 01
0000 46 1-001 - original
0000 47 1-002 - Change entry point name to BASSVAL L. JBS 02-MAY-1979
0000 48 1-003 - Add BASIC linkages for scaling. RW 26-JUN-79
0000 49 1-004 - Use new conversion routines. RW 9-JUL-79
0000 50 1-005 - Add an optional second argument to BASSVAL D. JBS 30-JUL-1979
0000 51 1-006 - Don't let conversion routine round for single precision. RW 20-Aug-79
0000 52 1-007 - Change bit set for integer ignore tabs. RW 30-Aug-79
0000 53 1-008 - Rechange bit set for integer ignore tabs. RW 31-Aug-79
0000 54 1-009 - KLUDGE!!!! WORKAROUND OTSCVTTL BUG. CHANGE CALL BACK. RW 7-SEPT-79
0000 55 1-010 - Remove kludge of edit 9. RW 11-Sept-79
0000 56 1-011 - String cleanup, don't use $STR$ macros. 30-Oct-79
0000 57 1-012 - Integerize after scaling. JBS 18-DEC-1979
```

```

0000 58 : 1-013 - Change MTH$DFLOOR to MTH$DINT. JBS 20-DEC-1979
0000 59 : 1-014 - Add support for g and h floating. PLL 25-Sep-81
0000 60 : 1-015 - Add support for packed decimal. PLL 8-Feb-82
0000 61 : 1-016 - Decimal entry point should check a flag in the frame before
0000 62 : calling the conversion routine. PLL 30-Jun-1982
0000 63 : 2-001 - Adapted from OTSS$CVTTR, version 1-010, from OTSS$CVTTIL,
0000 64 : version 1-007 and from BAS$VAL, version 1-007.
0000 65 : MDL 15-Jul-1982
0000 66 : 2-002 - use new routine OTSS$RET_A_CVT_TAB_R1 to get the address of the
0000 67 : convert table. make external ref's PIC. MDL 23-Jun-1983
0000 68 : 2-003 - minor bugfix in BAS$VAL_P. MDL 25-Jul-1983
0000 69 : 2-004 - BAS$VAL_D takes scale factor by VALUE, not by REF. MDL 8-Feb-1984
0000 70 : --

```

```
0000 72      .SBTTL  DECLARATIONS
0000 73
0000 74      :
0000 75      : INCLUDE FILES:
0000 76      :
0000 77      :
0000 78      :
0000 79      : EXTERNAL SYMBOLS:
0000 80      :
0000 81      .DSABL  GBL
0000 82      .EXTRN  BAS$HANDLER      : BASIC handler routine
0000 83      .EXTRN  BAS$K_ILLNUM     : illegal numeric input
0000 84      .EXTRN  OT$$$RET_A_CVT_TAB_R1 : Convert table address routine
0000 85      .EXTRN  BAS$CVT_T_P      : Convert text to packed routine
0000 86      .EXTRN  OT$$$CVT_MUL     : Conversion multiply routine
0000 87      .EXTRN  BAS$$STOP        : general purpose abort routine
0000 88      .EXTRN  BAS$$$SCALE_L_R1 : generates scale value
0000 89      .EXTRN  MTH$DINT         : integerization routine
0000 90
0000 91      :
0000 92      : MACROS:
0000 93      :
0000 94      :
0000 95      :
0000 96      : PSECT DECLARATIONS:
0000 97      :
0000 98
00000000 99      .PSECT  _BAS$CODE      PIC, SHR, LONG, EXE, NOWRT
0000 100
0000 101      :
0000 102      : EQUATED SYMBOLS:
0000 103      :
0000 104      :
0000 105      :+
0000 106      : WARNING !!!!!!!!!!!      WARNING !!!!!!!!!!!
0000 107      :
0000 108      : The following definitions are duplicated from the BLISS require
0000 109      : file BASFRAME.REQ.  If any changes are made, they MUST be duplicated
0000 110      : in both places!
0000 111      :-
0000000C 0000 112      BSF$A_SAVED_FP = 12      : saved Frame Pointer
FFFFF6E6 0000 113      BSF$W_FCD_FLAGS = -26    : flags longword in caller's frame
00000009 0000 114      BSF$M_FCD_RND  = 9      : "round" bit (in flags longword)
0000 115
0000 116      :+
0000 117      : argument pointer offsets
0000 118      :-
00000004 0000 119      string          = 4
0000 120
0000 121      :+
0000 122      : bits in flags longword passed to conversion routine
0000 123      :-
00000001 0000 124      ignore_blanks = 1
00000010 0000 125      ignore_tabs  = 16
00000008 0000 126      dont_round   = 8
0000 127
0000 128      :+
```



```
0000 129 : entry masks
0000 130 :-
00000FFC 0000 131 REGMASK = ^M< R2, R3, R4, R5, R6, R7, R8, R9, R10, R11 >
00000FF0 0000 132 REGMASK_H = ^M< R4, R5, R6, R7, R8, R9, R10, R11 >
0000 133 : register save mask
0000 134 : Note: integer overflow not enabled
0000 135
0000 136 :+
0000 137 : The following symbols are used to indicate the bit position of the flag
0000 138 : register.
0000 139 :-
0000 140
0000001F 0000 141 V_NEGATIVE = 31 : flag bit: 1 if negative sign
0000001E 0000 142 V_DEC_POINT = 30 : flag bit: 1 if decimal point is seen
40000000 0000 143 M_DEC_POINT = 1a30 : mask for V_DEC_POINT
0000001D 0000 144 V_NEG_DECEXP = 29 : flag bit: T if exponent has negative sign
20000000 0000 145 M_NEG_DECEXP = 1a29 : mask for V_NEG_DECEXP
0000001C 0000 146 V_DECEXP = 28 : flag bit: T if exponent field exist
10000000 0000 147 M_DECEXP = 1a28 : mask for V_DECEXP
0000001B 0000 148 V_EXT_BITS = 27 : flag bit: T if extension bits
0000 149 : wanted
08000000 0000 150 M_EXT_BITS = 1a27 : mask for V_EXT_BITS
0000 151
00000010 0000 152 V_DIGIT = 16 : flag bit: 1 if digit is seen
00010000 0000 153 M_DIGIT = 1a16 : mask for V_DIGIT
0000 154 :+
0000 155 : Literals for data types
0000 156 :-
00000000 0000 157 K_DTYPE_D = 0 : D-floating
00000001 0000 158 K_DTYPE_G = 1 : G-floating
00000002 0000 159 K_DTYPE_H = 2 : H-floating
00000003 0000 160 K_DTYPE_F = 3 : floating
0000 161
0000 162 :+
0000 163 : Temporary stack offsets
0000 164 :-
0000 165
00000000 0000 166 TEMP = 0 : temporary storage during
0000 167 : 8 word shift
00000004 0000 168 FLAG = 4 : flag storage
0000 169 : was R6 in FOR$CNV_IN_DEFG
00000008 0000 170 DIGITS = 8 : digits to right of decimal
0000 171 : point (was R7)
0000000C 0000 172 DECEXP = 12 : Decimal exponent
00000010 0000 173 DTYPE = 16 : Datatype code
0000 174
0000 175 :+
0000 176 : Stack offsets for OTS$$CVT_MUL routine
0000 177 :-
00000014 0000 178 BINNUM = 20 : Binary fraction storage
00000024 0000 179 INT = 36 : Overflow area for BINNUM
00000028 0000 180 BINEXP = 40 : Binary exponent
0000002C 0000 181 PRODF_4 = 44 : Multiply temporary
00000030 0000 182 PRODF = 48 : Multiply temporary
00000040 0000 183 CRY = 64 : Carry save area
00000050 0000 184 FRAME = CRY + 16 : Stack frame size
0000 185
```

00000000 186 :+
00000000 187 : Constants
00000000 188 :-
00000000 189
00000000 190 L_2P31_DIV_10 = 214748364 ; (2**31)/10
00000000 191


```
0000 193 .SBTTL BASSVAL_x - convert text to floating
0000 194
0000 195 :++
0000 196 : FUNCTIONAL DESCRIPTION:
0000 197 :
0000 198 : BASSVAL_x converts a text string containing a representation
0000 199 : of a numeric value to a floating representation of that
0000 200 : value.
0000 201 :
0000 202 : The description of the text representation converted by
0000 203 : BASSVAL_x is as follows:
0000 204 :
0000 205 : <0 or more blanks>
0000 206 : <'+' or '-' or nothing>
0000 207 : <0 or more decimal digits>
0000 208 : <'.' or nothing>
0000 209 : <0 or more decimal digits>
0000 210 : <exponent or nothing, where exponent is:
0000 211 : < <'E', 'e', 'D', 'd', 'Q', 'q'>
0000 212 : <0 or more blanks>
0000 213 : <'+' or '-' or nothing>>
0000 214 : or
0000 215 : <'+' or '-'>>
0000 216 : <0 or more decimal digits>>
0000 217 : <end of string>
0000 218 :
0000 219 : Notes:
0000 220 : the only valid exponent letters are
0000 221 : 'E' and 'e'; any others will be treated
0000 222 : as an invalid character.
0000 223 :
0000 224 : tab and blank characters are ignored.
0000 225 :
0000 226 : exponent must start with a valid exponent letter.
0000 227 :
0000 228 :
0000 229 : CALLING SEQUENCE:
0000 230 :
0000 231 : value.wlc.v = BASSVAL_x (in_str.rt.dx [, scale_factor.rl.v])
0000 232 :
0000 233 : where 'x' is the datatype of the floating value, either
0000 234 : F, D, G or H.
0000 235 :
0000 236 :
0000 237 : INPUT PARAMETERS:
0000 238 :
0000 239 : in_str = 4 ; input string descriptor by
0000 240 : ; reference.
0000 241 : scale_val = 8 ; optional scale value
0000 242 :
0000 243 :
0000 244 : IMPLICIT INPUTS:
0000 245 :
0000 246 :
0000 247 : NONE
0000 248 :
0000 249 : OUTPUT PARAMETERS:
```

```
00000008 0000 250 ;
00000018 0000 251 ; value = 8
0000 0000 252 ; ext_bits = 24
0000 0000 253 ; floating result by ref
0000 0000 254 ; If present, the value will
0000 0000 255 ; NOT be rounded and the first
0000 0000 256 ; n bits after truncation will
0000 0000 257 ; be returned in this argument.
0000 0000 258 ; For D-floating, the next 8 bits
0000 0000 259 ; are returned as a byte.
0000 0000 260 ; For G and H floating, 11 and 15
0000 0000 261 ; bits are returned, respectively,
0000 0000 262 ; as a word, left-adjusted.
0000 0000 263 ; These values are suitable for
0000 0000 264 ; use as the extension operand
0000 0000 265 ; in an EMOD instruction.
0000 0000 266 ; WARNING: The bits returned for
0000 0000 267 ; H-floating may not be precise,
0000 0000 268 ; due to the fact that calculations
0000 0000 269 ; are only carried to 128 bits.
0000 0000 270 ; However, the error should be
0000 0000 271 ; small. D and G datatypes
0000 0000 272 ; return guaranteed exact bits,
0000 0000 273 ; but they are not rounded.
0000 0000 274 ;
0000 0000 275 ; IMPLICIT OUTPUTS:
0000 0000 276 ; NONE
0000 0000 277 ;
0000 0000 278 ; COMPLETION CODES:
0000 0000 279 ;
0000 0000 280 ; BASSK_ILLNUM - Error if illegal character in input or
0000 0000 281 ; overflow.
0000 0000 282 ; SS$_NORMAL - success
0000 0000 283 ;
0000 0000 284 ; SIDE EFFECTS:
0000 0000 285 ; NONE
0000 0000 286 ;
0000 0000 287 ;
0000 0000 288 ; --
0000 0000 289 ;
0000 0000 290 ;
0000 0000 291 ;
0000 0000 292 ; .ENTRY BASSVAL_H, REGMASK_H
0002 0002 293 ; entry for BASSVAL_H
SE 00000050 8F C2 0002 294 ; Create stack frame
10 AE 02 D0 0009 295 ; Set datatype code
45 11 000D 296 ; Go to common code
000F 297 ;
000F 298 ; .ENTRY BASSVAL_G, REGMASK
0011 299 ; entry for BASSVAL_G
SE 00000050 8F C2 0011 300 ; Create stack frame
10 AE 01 D0 0018 301 ; Set datatype code
36 11 001C 302 ; Go to common code
001E 303 ;
001E 304 ; .ENTRY BASSVAL_F, REGMASK
0020 305 ; entry for BASSVAL_F
SE 00000050 8F C2 0020 306 ; Create stack frame
```

```
10 AE 03 D0 0027 307      MOVL  #K DTYPE_F, DTYPE(SP) ; Set datatype code
      27 11 0028 308      BRB    COMMON                ; Go to common code
      OFFC 002D 309
      002F 310      .ENTRY BASSVAL_D, REGMASK
SE 00000050 8F C2 002F 311      ; entry for BASSVAL_D
      10 AE 00 D0 0036 312      SUBL2 #FRAME, SP          ; Create stack frame
      02 6C 91 003A 313      MOVL  #K DTYPE_D, DTYPE(SP) ; Set datatype code
      06 1F 003D 314      CMPB   (AP), #2                ; Optional scale value present?
      5A 08 AC D0 003F 315      BLSSU MAKE_VALUE          ; no, make one up
      OF 11 0043 316      MOVL  scale_val(AP), R10        ; yes, get it
      0045 317      BRB    COMMON
      0045 318
      0045 319      MAKE_VALUE:
      0045 320      MOVL  BSFSA SAVED_FP(FP), R0          ; get scale factor from caller's frame
      0049 321      JSB    G^BASSSCALE_L_R1              ; determine scale value (place in R0)
      004F 322      MOVL  R0, R10                        ; put it where it belongs
      0052 323      BRB    COMMON
      0054 324
      0054 325      ;+
      0054 326
      0054 327      Register usage and abbreviations:
      0054 328      R0 - Generally count of input characters remaining.
      0054 329      R1 - Generally pointer to input character.
      0054 330      R2 - Generally holds decimal exponent.
      0054 331      R3 - Used first to hold current character, then as
      0054 332      extra precision bits for the fraction.
      0054 333      R4-R7 - The 128 bit binary fraction.
      0054 334      R8 - Count of digits seen after overflow.
      0054 335      R9 - Count of significant digits seen in fraction (number of
      0054 336      digits currently held in R4:R7).
      0054 337      R10 - optional scale value.
      0054 338
      0054 339      FAC: Binary fraction, R4-R7.
      0054 340      :-
      0054 341
      0054 342      COMMON:
      04 AE D4 0054 343      CLRL  FLAG(SP)                ; clear flags
      50 04 BC 7D 0057 344      5$: MOVQ  @in_str(AP), R0    ; R0 will get string length, the
      0058 345      ; CLASS and TYPE fields will go
      0058 346      ; away after the first SKPC.
      0058 347      ; R1 points to input string.
      52 D4 0058 348      CLRL  R2                        ; R2 = DECIMAL_EXPONENT = 0
      54 7C 005D 349      CLRQ  R4                        ; R4-R7 = FAC = 0
      56 7C 005F 350      CLRQ  R6
      08 AE D4 0061 351      CLRL  DIGITS(SP)              ; digits in fraction
      0064 352
      58 7C 0064 353      10$: CLRQ  R8                    ; Clear digit counts (R8 & R9).
      0066 354
      0066 355
```



```
0066 357 :+
0066 358 : find first non-blank. If none, return zero. Otherwise process
0066 359 : character.
0066 360 :-
0066 361
61 50 20 3B 0066 362 20$: SKPC #A/ /, R0, (R1) : skip blanks
006A 363 : R0 = #CHAR REMAINING
006A 364 : R1 = POINTER_TO_INPUT
006A 365 : Z bit is set if all blanks
006A 366 : nor-blank found?
00E6 31 006C 367 BGTR 30$ BRW ZERO : if not, return zero
53 61 9A 006F 368 30$: MOVZBL (R1), R3 : R3 = ASCII(current_char)
09 53 D1 0072 369 : CMPL R3, #9 : Is character a tab?
08 12 0075 370 : BNEQ 35$ : No
51 D6 0077 371 : INCL R1 : Yes, bump pointer
EA 50 F5 0079 372 : SOBGTR R0, 20$ : Decrement character count
00D6 31 007C 373 : BRW ZERO : Value is zero
2D 53 91 007F 374 35$: CMPB R3, #A/-/ : is current char a "-" sign?
05 12 0082 375 : BNEQ 40$ : branch if not
15 04 AE 1F E3 0084 376 : BBBS #V_NEGATIVE, FLAG(SP), DIGIT_LOOP
0089 377 : set negative flag and continue
28 53 91 0089 378 40$: CMPB R3, #A/+/: : is current char a "+" sign?
10 13 008C 379 : BEQL DIGIT_LOOP : yes, ignore and continue
2E 53 91 008E 380 : CMPB R3, #A/./ : is current char a "."?
15 12 0091 381 : BNEQ CHECK_DIGIT : no, should be a digit
04 AE 40000000 8F C8 0093 382 : BISL #M_DEC_POINT, FLAG(SP) : set decimal point encountered
08 AE D4 009B 383 : CLRL DIGITSP : ignore digits_in_fract
009E 384
```

```

009E 386 :+
009E 387 : Collect integer and fraction digits. Blanks and tabs are ignored.
009E 388 :-
009E 389
009E 390 DIGIT_LOOP:
031A 30 009E 391 BSBW RGET : get a new character
50 D5 00A1 392 TSTL R0 : check for end of string
03 14 00A3 393 BGTR CHECK_DIGIT : continue if positive
00A9 31 00A5 394 BRW SCALE : done if string empty
00A8 395 CHECK_DIGIT:
53 30 C2 00A8 396 SUBL #A/0/, R3 : convert to numeric
09 53 D1 00AB 397 CMPL R3, #9 : is it a digit?
22 1A 00AE 398 BGTRU NOT DIGIT : no
04 AE 00010000 8F C8 00B0 399 BISL #M DIGIT, FLAG(SP) : yes, set digit encountered
OCCCCCCC 8F 57 D1 00B8 400 CMPL R7, #L_2P31_DIV_10 : check highest part of FAC to
: see if it is too big to
: multiply by 10.
04 18 00BF 401 : it's ok
58 D6 00C1 402 BLEQU 10$ : overflow, bump counter
03 11 00C3 403 INCL R8 : skip multiplication
0309 30 00C5 404 BRB 2$ : skip multiplication
D1 04 AE 1E E1 00C8 405 10$: BSBW MUL10 R9 : Multiply FAC by 10 and add R3.
: DIGIT_LOOP
00CD 406 2$: BBC #V_DEC_POINT, FLAG(SP), : check to see if decimal
: point has been seen
: - continue if not.
00CD 407 : bump DIGITS
00CD 408 : branch back to read more
08 AE D6 00CD 409 INCL DIGITS(SP)
CC 11 00D0 410 BRB DIGIT_LOOP
00D2 411
00D2 412
00D2 413

```

```
00D2 415 :+
00D2 416 : A non-digit has been found. Check for decimal point or exponent letter.
00D2 417 :-
00D2 418
00D2 419 NOT_DIGIT:
FFFFF8F 8F 53 D1 00D2 420 CMPL R3, #<^A/. /-^A/O/> : check if current char is a "."
15 0D 13 00D9 421 BEQL DECIMAL_POINT : branch to DECIMAL_POINT if yes
35 53 D1 00DB 422 CMPL R3, #<^A/E /-^A/O/> : "E"?
16 13 00DE 423 BEQL EXPON : process exponent
35 53 D1 00E0 424 CMPL R3, #<^A/e /-^A/O/> : "e"?
11 13 00E3 425 BEQL EXPON : process exponent
008B 31 00E5 426 BRW ERROR : none of the above => ERROR.
00E8 427
00E8 428 :+
00E8 429 : Decimal point has been found
00E8 430 :-
00E8 431
00E8 432 DECIMAL_POINT:
06 04 AE 1E E2 00E8 433 BBSS #V DEC POINT, FLAG(SP), 10$ : error if duplicate
08 AE D4 00ED 434 CLRL DIGITS(SP) : reset DIGITS
FFAB 31 00F0 435 BRW DIGIT_LOOP : get fraction digits
007D 31 00F3 436 10$: BRW ERROR
```



```
00F6 438 :+
00F6 439 : we have an exponent. see if we have gotten any digits yet; if we
00F6 440 : haven't, this is an error.
00F6 441 :-
00F6 442 EXPON:
03 04 AE 10 E0 00F6 443 BBS #V DIGIT, FLAG(SP), EXPON_DIGITS
0075 31 00FB 444 BRW ERROR ; if digit seen bit not set, error
00FE 445
00FE 446 :+
00FE 447 : Loop to collect digits, store the accumulated DECIMAL_EXPONENT in R2
00FE 448 :-
00FE 449 EXPON_DIGITS:
50 D7 00FE 450 DECL R0 ; skip over letter
3F 15 0100 451 BLEQ EXP_DONE ; done if string empty
51 D6 0102 452 INCL R1 ; R1 points to next character
61 50 20 3B 0104 453 SKPC #^A/ /, R0, (R1) ; skip blanks
37 15 0108 454 BLEQ EXP_DONE ; done if end of string
53 61 9A 010A 455 MOVZBL (R1), R3 ; R3 = current char
09 53 D1 010D 456 CMPL R3, #9 ; Is it a tab?
EC 13 0110 457 BEQL EXPON_DIGITS ; Yes, skip it
2B 53 D1 0112 458 10$: CMPL R3, #^A/+ / ; "+"?
OD 13 0115 459 BEQL EXP_LOOP ; yes, get digits
2D 53 D1 0117 460 CMPL R3, #^A/- / ; "-"?
OF 12 011A 461 BNEQ EXP_CHECK ; no, go check digit
04 AE 20000000 8F C8 011C 462 EXP_NEG: BISL #M_NEG_DECEXP, FLAG(SP) ; exponent is negative
0124 463 EXP_LOOP: BSBW RGET ; get next character
0294 30 0124 464 TSTL R0 ; is string empty?
50 D5 0127 465 BLEQ EXP_DONE ; done if true
16 15 0129 466 EXP_CHECK: SUBL #^A/0/, R3 ; convert to numeric
53 30 C2 012B 467 BLSS ERROR ; If negative, illegal character
43 19 012E 470 CMPL R3, #9 ; is it a digit?
09 53 D1 0130 471 BGTRU ERROR ; branch to ERROR if not
3E 1A 0133 472 MULL #10, R2 ; add in new digit
52 0A C4 0135 473 BVS ERROR ; overflow?
39 1D 0138 474 ADDL R3, R2 ; to exponent
52 53 C0 013A 475 BVS ERROR ; overflow?
34 1D 013D 476 BRB EXP_LOOP ; get more exponent digits
E3 11 013F 477
0141 478
0141 479 EXP_DONE: BBC #V_NEG_DECEXP, FLAG(SP), 1$ ; check for negative
03 04 AE 1D E1 0141 480 MNEGL R2, R2 ; negate DECIMAL_EXPONENT
52 52 CE 0146 481 1$: BISL #M_DECEXP, FLAG(SP) ; exponent field exists
04 AE 10000000 8F C8 0149 482
0151 483
0151 484
```

```
0151 486 ;+
0151 487 ; Done collecting input characters for digits and/or exponent
0151 488 ; If FAC=0, no scaling is necessary, just store 0.0 and return.
0151 489 ; -
0151 490
0151 491 SCALE:
59 05 0151 492 TSTL R9 ; Check FAC for zero.
2B 12 0153 493 BNEQ INIT_BINEXP ; Branch if not.
0155 494
0155 495 ;+
0155 496 ; Value is zero.
0155 497 ; -
0155 498
0155 499 ZERO:
03 00 10 AE 8F 0155 500 CASEB DTYPE(SP), #K_DTYPE_D, #K_DTYPE_F ; Select on datatype
0008' 015A 501 1$: .WORD D_NUM-1$
000C' 015C 502 .WORD G_NUM-1$
0010' 015E 503 .WORD H_NUM-1$
0016' 0160 504 .WORD F_NUM-1$
0162 505 D_NUM:
50 7C 0162 506 CLRQ R0
0C 11 0164 507 BRB ZERO_RET
0166 508 G_NUM:
50 7C 0166 509 CLRQ R0
08 11 0168 510 BRB ZERO_RET
016A 511 H_NUM:
50 7C 016A 512 CLRQ R0 ; zero out return value
52 7C 016C 513 CLRQ R2
02 11 016E 514 BRB ZERO_RET
50 D4 0170 515 F_NUM:
516 CLRL R0
517 BRW ZERO_RET
0172 518
0172 519 ZERO_RET:
04 0172 520 RET ; return.
0173 521
0173 522 ;+
0173 523 ; ERROR return
0173 524 ; -
0173 525
0173 526 ERROR:
7E 00'8F 9A 0173 527 MOVZBL #BAS$K_ILLNUM, -(SP) ;
00000000'GF 01 FB 0177 528 CALLS #1, G^BAS$$STOP
D5 11 017E 529 BRB ZERO ; Set value to zero and exit
0180 530
0180 531 ;+
0180 532 ; Set R1 to the binary exponent [exponent bias + 128 - 1].
0180 533 ; 128 is number of fraction bits and 1 is
0180 534 ; for the MSB fraction bit which will be hidden later.
0180 535 ; BINARY_EXPONENT will be modified during normalization process.
0180 536 ; -
0180 537
0180 538 INIT_BINEXP:
03 00 10 AE 8F 0180 539 CASEB DTYPE(SP), #K_DTYPE_D, #K_DTYPE_F ; Select on datatype
0008' 0185 540 1$: .WORD D_EXP-1$
000F' 0187 541 .WORD G_EXP-1$
0016' 0189 542 .WORD H_EXP-1$
```

```
51 00FF 8F 3C 018B 543 .WORD F_EXP-1$
15 11 018D 544 D_EXP: MOVZWL #2^X80+^X7F>, R1 ; D-Floating
51 047F 8F 3C 0192 545 BRB EXP_COMMON
0E 11 0194 546 G_EXP: MOVZWL #<^X400+^X7F>, R1 ; G-Floating
51 407F 8F 3C 0199 547 BRB EXP_COMMON
07 11 019B 548 H_EXP: MOVZWL #<^X4000+^X7F>, R1 ; H-Floating
51 00FF 8F 3C 01A0 549 BRB EXP_COMMON
00 11 01A2 550 F_EXP:
01A2 551 MOVZWL #<^X80+^X7F>, R1
01A7 552 BRB EXP_COMMON
01A9 553
01A9 554 ;+
01A9 555 ; Find the true decimal exponent for the value expressed in FAC.
01A9 556 ; True decimal exponent = Explicit exponent - [scale factor] -
01A9 557 ; digits in fraction + number of overflows
01A9 558 ; -
01A9 559
01A9 560 EXP_COMMON:
50 52 D0 01A9 561 MOVL R2, R0 ; R0 = DECIMAL_EXPONENT
02 6C 91 01AC 562
03 1F 01AF 563 CMPB (AP), #2 ; optional scale factor present?
50 5A C2 01B1 564 BLSSU 20$ ; no
01B4 565 SUBL R10, R0 ; yes, adjust decimal exponent for
58 08 AE C2 01B4 566 20$: SUBL DIGITS(SP), R8 ; scale factor
01B8 567 ; adjust for digits in fraction
OC AE 50 58 C1 01B8 568
B4 1D 01BD 569 ADDL3 R8, R0, DECEXP(SP) ; adjust decimal exponent for overflow
01BF 570 BVS ERROR ; if overflow, error
571
```



```
01BF 573 :+
01BF 574 : Normalization. Shift the value left until bit 31 of R7 is on.
01BF 575 : Adjust the binary exponent appropriately.
01BF 576 :-
01BF 577 :
09 59 D1 01BF 578 CMPL R9, #9 : Are there more than 9 digits?
35 15 01C2 579 BLEQ N1 : If not, use N1.
12 59 D1 01C4 580 CMPL R9, #18 : Are there more than 18 digits?
1A 15 01C7 581 BLEQ N2 : If not, use N2.
01C9 582 :+
01C9 583 : Process all four longwords, since there are more than 18 digits.
01C9 584 :-
6E 40 57 1F E0 01C9 585 N4: BBS #31, R7, REBASE : Quit when R7<31> = 1.
55 01 1F EF 01CD 586 EXTZV #31, #1, R5, TEMP(SP) : Save bit lost in shift.
54 54 01 79 01D2 587 ASHQ #1, R4, R4 : Shift low part by one bit.
56 56 01 79 01D6 588 ASHQ #1, R6, R6 : Shift high part by one bit.
56 01 00 6E F0 01DA 589 INSV TEMP(SP), #0, #1, R6 : Replace bit lost in shift.
51 51 D7 01DF 590 DECL R1 : Adjust exponent by one.
E6 11 01E1 591 BRB N4 : Go back and retest.
01E3 592 :+
01E3 593 : Process two low-order longwords only, since there are <= 18 digits.
01E3 594 :-
51 00000040 8F C2 01E3 595 N2: SUBL #64, R1 : Adjust exponent by 64.
56 54 7D 01EA 596 MOVQ R4, R6 : "Shift" by 64 bits.
51 D7 01ED 597 10$: DECL R1 : Adjust exponent by one.
56 56 01 79 01EF 598 ASHQ #1, R6, R6 : Shift one bit.
F8 18 01F3 599 BGEQ 10$ : If R7<31> = 0, repeat.
54 7C 01F5 600 CLRQ R4 : Clear low-order 64 bits.
14 11 01F7 601 BRB REBASE : Continue with next phase.
01F9 602 :+
01F9 603 : Process only the low-order longword, since there are <= 9 digits.
01F9 604 :-
51 00000060 8F C2 01F9 605 N1: SUBL #96, R1 : Adjust exponent by 96.
57 54 D0 0200 606 MOVL R4, R7 : "Shift" by 96 bits.
51 D7 0203 607 20$: DECL R1 : Adjust exponent.
57 57 01 78 0205 608 ASHL #1, R7, R7 : Shift one bit.
F8 18 0209 609 BGEQ 20$ : If R7<31> = 0, repeat.
54 D4 020B 610 CLRL R4 : Clear low-order longword.
020D 611 :+
020D 612 : Rebasing. R4-R7 now contains a binary fraction normalized with
020D 613 : the radix point to the left of bit 31 of R7. R1 contains the
020D 614 : current binary exponent and DECEXP(SP) contains the current decimal
020D 615 : exponent.
020D 616 :
020D 617 : Therefore, the number can be represented as:
020D 618 :  $2^{**b} * \text{fraction} * 10^{**d}$ 
020D 619 : where b is the binary exponent and d is the decimal exponent. We
020D 620 : call OTSS$CVT_MUL to multiply the number by some power of 10 such
020D 621 : that d goes to zero and b goes to the appropriate value. When d is
020D 622 : zero, b contains the proper binary exponent.
020D 623 :
020D 624 :-
020D 625 :
020D 626 REBASE:
58 14 AE 9E 020D 627 MOVAB BINNUM(SP), R8 : R8 is used by subroutine as base
28 AE 51 D0 0211 628 MOVL R1, BINEXP(SP) : Store binary exponent
14 AE 54 7D 0215 629 MOVQ R4, BINNUM+0(SP) : Store fraction
```

```

; Convert text to numeric
BAS$VAL_x - convert text to floating
16-SEP-1984 00:01:37 VAX/VMS Macro V04-00
6-SEP-1984 10:39:35 [BASRTL.SRC]BASVAL.MAR;1

1C AE 56 7D 0219 630 MOVQ R6, BINNUM+8(SP)
57 0D D0 021D 631 MOVL #15, R7 ; Highest bit number possibly
; on in decimal exponent.
52 14 D0 0220 632 10$: MOVL #20, R2 ; Initially, positive offset
50 0C AE D0 0223 633 MOVL DECEXP(SP), R0 ; Get decimal exponent
41 13 0227 634 BEQL FLOAT ; If zero, we're done
06 14 0229 635 BGTR 20$ ; Positive?
52 14 CE 022B 636 MNEGL #20, R2 ; No, use negative offset
50 50 CE 022E 637 MNEGL R0, R0 ; Absolute value
10 50 D1 0231 638 20$: CMPL R0, #16 ; Within linear table range?
0B 15 0234 639 BLEQ 50$ ; Yes
03 50 57 E0 0236 640 30$: BBS R7, R0, 40$ ; Is the R7th bit of R0 on?
F9 57 F4 023A 641 SOBGEQ R7, 30$ ; No, try again.
; This can never fall through.
50 57 0C C1 023D 642 40$: ADDL3 #12, R7, R0 ; Index is 12+bit position
; because table is linear
; from 0-16.
52 50 C4 0241 643 50$: MULL2 R0, R2 ; Get table offset
00000000 GF 16 0244 644 JSB G*OT$$$RET_A_CVT_TAB_R1 ; get convert table address (in R0)
52 50 C0 024A 645 ADDL2 R0, R2 ; Table entry address
6E 57 D0 024D 646 MOVL R7, TEMP(SP) ; Save hi bit position
57 28 AE 9E 0250 647 MOVAB DECEXP+28(SP), R7 ; This is "common convert routine"
; table base. The +28 offsets
; the -28 location of DEC_EXP
; referenced in OT$$$CVT_MUL.
00000000 GF 16 0254 648 JSB G*OT$$$CVT_MUL ; Do the multiplication
57 6E 01 C3 025A 649 SUBL3 #1, TEMP(SP), R7 ; Get next bit position
C0 18 025E 650 BGEG 10$ ; Loop back if more
0260 651
0260 652
0260 653
0260 654
0260 655 JSB G*OT$$$CVT_MUL
57 6E 01 C3 025A 656 SUBL3 #1, TEMP(SP), R7
C0 18 025E 657 BGEG 10$
0260 658
0260 659 ;+
0260 660 ; If we fall through here, then there are no more bits to reduce.
0260 661 ; Test DECEXP to make sure.
0260 662 ; -
0260 663
0C AE D5 0260 664 TSTL DECEXP(SP) ; Any bits still on?
05 13 0263 665 BEQL FLOAT ; No, ok
15 19 0265 666 BLSS UNDERFLOW ; Negative, underflow
FF09 31 0267 667 BRW ERROR ; Yes, exponent too big

```

```
026A 669 :+
026A 670 : Create a floating number from the fraction in BINNUM and the
026A 671 : binary exponent in R1. Each datatype has a separate routine
026A 672 : to do this.
026A 673 :-
026A 674
026A 675
026A 676 FLOAT:
026A 677 TSTL BINEXP(SP) : Underflow?
026D 678 BLSS UNDERFLOW : Yes
026F 679 CASEB DTYPE(SP), #K_DTYPE_D, #K_DTYPE_F
0274 680 10$: .WORD FLOAT_D-10$
0276 681 .WORD FLOAT_G-10$
0278 682 .WORD FLOAT_H-10$
027A 683 .WORD FLOAT_F-10$
027C 684
027C 685 :+
027C 686 : Value underflowed. Set to zero.
027C 687 :-
027C 688
027C 689 UNDERFLOW:
027C 690 BRW ZERO
027F 691
```

03 00 28 AE D5 026A 677
OD 19 026D 678
10 AE 8F 026F 679
000B' 0274 680
0062' 0276 681
00A7' 0278 682
01J7' 027A 683
FED6 31 027C 688
027C 689
027C 690
027F 691


```
51 56 1C AE 7D 027F 693 FLOAT_D:
28 AE 17 78 027F 694 MOVQ BINNUM+8(SP), R6 : Restore fraction
58 56 49 1D 0283 695 ASHL #23, BINEXP(SP), R1 : Put exponent in proper place
56 56 F8 8F 9A 0288 696 BVS ERROR_D : Error if overflows
57 FF000000 8F 79 028A 697 MOVZBL R6, R8 : Extract rounding bits
57 57 51 C0 028D 698 ASHQ #-8, R6, R6 : Shift fraction right 8 places
57 57 35 1D 0292 699 BICL #~XFF000000, R7 : clear possibly shifted bits
07 58 07 E1 0299 700 ADDL R1, R7 : Add in exponent
57 57 56 D6 029C 701 BVS ERROR_D : overflow if hidden bit bumps
07 58 07 E1 029E 702 : exponent too far
57 57 00 D8 02A2 703 BBC #7, R8, 15$ : round bit is zero
04 04 AE 1B E1 02A4 704 INCL R6 : round
04 04 BC 58 90 02A7 705 ADWC #0, R7
04 04 AE 1F E1 02A9 706 BVS ERROR_D : Error?
00 57 1F E3 02AE 707 15$: BBC #V_EXT_BITS, FLAG(SP), 17$
50 57 10 9C 02B2 708 MOV B R8, @ext_bits(AP)
51 56 10 9C 02B7 709 17$: BBC #V_NEGATIVE, FLAG(SP), 20$ : Set sign bit
5A D5 02BB 710 BBCS #3T, R7, 20$ : insert sign bit to 1
09 13 02BB 711 20$: ROTL #16, R7, R0 : rotate and store result
5E DU 02BF 712 ROTL #16, R6, R1
01 FB 02C3 713 TSTL R10 : scale factor > 0 ?
00E7 31 02C5 714 BEQL 25$ : no, return raw result
FE9D 31 02C7 715 PUSH SP :
02C9 716 CALLS #1, G*MTN$DINT : integerize the result
02D0 717 BRW EXIT : All done
02D3 720 25$:
02D3 721 ERROR_D:
02D3 722 BRW ERROR : error return
02D6 723
724
```

```

      51 56 1C AE 7D 02D6 726 FLOAT_G:
      28 AE 14 78 02D6 727      MOVQ  BINNUM+8(SP), R6      ; Restore fraction
      37 1D 78 02DA 728      ASHL  #20, BINEXP(SP), R1      ; Put exponent in proper place
58 56 0B 00 EF 02DF 729      BVS   ERROR_G                  ; Error if overflows
      58 58 05 9C 02E1 730      EXTZV #0, #11, R6, R8        ; Extract rounding bits
      56 56 F5 8F 79 02E6 731      ROTL  #5, R8, R8          ; Left adjust
57 FFE00000 8F 79 02EA 732      ASHQ  #-11, R6, R6          ; Shift fraction right 11 places
      57 57 51 1D 79 02EF 733      BICL  #*XFFE00000, R7      ; clear possibly shifted bits
      1D 1D 79 02F6 734      ADDL  R1, R7                    ; Add in exponent
      1D 1D 79 02F9 735      BVS   ERROR_G                  ; overflow if hidden bit bumps
      1D 1D 79 02FB 736      BVS   ERROR_G                  ; exponent too far
04 04 AE 1B E1 02FB 737 15$: BBC  #V_EXT_BITS, FLAG(SP), 17$ ;
      18 BC 58 B0 0300 738      MOVW  R8, @ext_bits(AP)      ;
04 04 AE 1F E1 0304 739 17$: BBC  #V_NEGATIVE, FLAG(SP), 20$ ; Set sign bit
      00 57 1F E3 0309 740      BBCS  #3T, R7, 20$          ; insert sign bit to 1
      50 57 10 9C 030D 741 20$: ROTL  #16, R7, R0          ; rotate and store result
      51 56 10 9C 0311 742      ROTL  #16, R6, R1          ;
      00A2 31 0315 743      BRW   EXIT                      ; All done
      F58 31 0318 744      BRW   EXIT                      ;
      F58 31 0318 745      BRW   EXIT                      ;
      F58 31 0318 746 ERROR_G: BRW   ERROR                  ; error return
      F58 31 0318 747
      F58 31 0318 748
```

```

      54 14 AE 7D 031B 750 FLOAT_H:
      56 1C AE 7D 031B 751      MOVQ BINNUM+0(SP), R4      ; Restore fraction
51 28 AE 10 78 0323 752      MOVQ BINNUM+8(SP), R6
      4E 1D 0328 753      ASHL #16, BINEXP(SP), R1      ; Step 1
58 54 OF 00 EF 032A 754      BVS ERROR_H      ; Error if overflows
      01 9C 032F 755      EXTZV #0, #15, R4, R8      ; Extract rounding bits
50 56 OF 00 EF 0333 756      ROTL #1, R8, R8      ; Left adjust
      F1 BF 79 0338 757      EXTZV #0, #15, R6, R0      ; shift right 15 places
      F1 BF 79 033D 758      ASHQ #-15, R4, R4
      F1 BF 79 033D 759      ASHQ #-15, R6, R6
55 56 OF 11 50 F0 0342 760      INSV R0, #17, #15, R5
57 FFFE0000 BF CA 0347 761      BICL #~XFFE0000, R7      ; clear possibly shifted bits
      51 51 CO 034E 762      ADDL R1, R7      ; Step 3
      25 1D 0351 763      BVS ERROR_H      ; overflow if hidden bit bumps
      0353 764      exponent too far
04 04 AE 1B E1 0353 765 15$: BBC #V_EXT_BITS, FLAG(SP), 17$
      18 BC 58 B0 0358 766      MOVW R8, @ext_bits(AP)
04 04 AE 1F E1 035C 767 17$: BBC #V_NEGATIVE, FLAG(SP), 20$      ; Step 4
      00 57 1F E3 0361 768      BBCS #3T, R7, 20$      ; insert sign bit to 1
      0365 769 20$:
50 57 10 9C 0365 770      ROTL #16, R7, R0      ; rotate and store result
51 56 10 9C 0369 771      ROTL #16, R6, R1
52 55 10 9C 036D 772      ROTL #16, R5, R2
53 54 10 9C 0371 773      ROTL #16, R4, R3
      0042 31 0375 774      BRW EXIT
      0378 775
      0378 776 ERROR_H:
FDF8 31 0378 777      BRW ERROR      ; error return
      0378 778
```



```
0388 812 .SBTTL RGET - get next character
0388 813
0388 814 :+
0388 815 : Subroutine RGET
0388 816 : input:
0388 817 : R0 = number of characters remaining in string
0388 818 : R1 = address of current character
0388 819 : output:
0388 820 : R0 is decremented by 1. If R0 is now non-positive,
0388 821 : RGET returns immediately, indicating that the end
0388 822 : of the string has been reached.
0388 823 : If there is string remaining, R1 now points to the
0388 824 : new current character, and R3 has that character.
0388 825 :
0388 826 :-
0388 827
0388 828 RGET:
50 50 0388 829 DECL R0 ; decrement length counter
11 15 038D 830 BLEQ 20$ ; If string empty, return
51 D6 038F 831 INCL R1 ; R1 points to new character
53 61 9A 03C1 832 MOVZBL (R1), R3 ; R3 gets character
09 53 D1 03C4 833 CMPL R3, #9 ; Is it a tab?
F2 13 03C7 834 BEQL RGET ; Yes
20 53 D1 03C9 835 10$: CMPL R3, #^A/ / ; is character a blank?
02 12 03CC 836 BNEQ 20$ ; return if not
EB 11 03CE 837 BRB RGET ; yes
05 03D0 838 20$: RSB ; return
```

```
.SBTTL MUL10_R9 - multiply FAC by 10 and add digit in R3
03D1 840
03D1 841
03D1 842
03D1 843 :+ Subroutine MUL10_R9
03D1 844 : input:
03D1 845 : R4-R7 - FAC
03D1 846 : R9 - count of decimal digits currently held in FAC
03D1 847 : output:
03D1 848 : R4-R7 - FAC*10 + digit in R3
03D1 849 : R9 - updated count
03D1 850 :-
03D1 851
03D1 852 MUL10_R9:
5C 59 09 F3 03D1 853 AOBLEQ #9, R9, M1 : If 9 or fewer digits, use M1.
12 59 D1 03D5 854 CMPL R9, #18 : If 18 or fewer digits,
40 15 03D8 855 BLEQ M2 : use M2.
03DA 856 :+
03DA 857 : Process entire octaword (four longwords), since there are > 18 digits.
03DA 858 :-
50 55 01 1F DD 03DA 859 M4: PUSHL R0 : Free up a scratch register.
56 56 01 79 03DC 860 EXTZV #31, #1, R5, R0 : Save bit that will be lost.
56 56 50 C0 03E1 861 ASHQ #1, R6, R6 : Multiply high part by 2.
54 54 01 79 03E5 862 ADDL R0, R6 : Replace bit lost in shift.
55 55 02 1E EF 03E8 863 ASHQ #1, R4, R4 : Multiply low part by 2.
7E 56 02 79 03EC 864 EXTZV #30, #2, R5, R0 : Save bits that will be lost.
6E 50 C0 03F1 865 ASHQ #2, R6, -(SP) : Multiply high part by 4.
7E 54 02 79 03F5 866 ADDL R0, (SP) : Replace bits lost in shift.
54 8E C0 03F8 867 ASHQ #2, R4, -(SP) : Multiply low part by 4.
55 8E D8 03FC 868 ADDL (SP)+, R4 : Add 8*FAC to 2*FAC.
56 8E D8 03FF 869 ADWC (SP)+, R5 : ...
57 8E D8 0402 870 ADWC (SP)+, R6 : ...
54 53 C0 0405 871 ADWC (SP)+, R7 : ...
55 09 1E 0408 872 ADDL R3, R4 : Add digit in R3.
56 00 D8 040B 873 BCC 20$ : If no carry, quit now.
57 00 D8 040D 874 ADWC #0, R5 : ...
50 8E D0 0410 875 ADWC #0, R6 : ...
50 8E D0 0413 876 ADWC #0, R7 : ...
50 8E D0 0416 877 20$: MOVL (SP)+, R0 : Restore scratch register.
05 0419 878 RSB : Return to caller.
041A 879 :+
041A 880 : Process two low-order longwords only, since there are <= 18 digits.
041A 881 :-
54 54 01 79 041A 882 M2: ASHQ #1, R4, R4 : Multiply R4:R5 by 2.
56 54 02 79 041E 883 ASHQ #2, R4, R6 : Multiply R4:R5 by 4.
54 56 C0 0422 884 ADDL R6, R4 : Add 8*FAC to 2*FAC (low).
55 57 D8 0425 885 ADWC R7, R5 : Add 8*FAC to 2*FAC (high).
54 53 C0 0428 886 ADDL R3, R4 : Add digit in R3.
55 00 D8 042B 887 ADWC #0, R5 : ...
56 7C 042E 888 CLRQ R6 : Restore R6:R7.
05 0430 889 RSB : Return to caller.
0431 890 :+
0431 891 : Process low-order longword only, since there are 9 or fewer digits.
0431 892 :-
54 6444 DE 0431 893 M1: MOVAL (R4)[R4], R4 : Multiply R4 by 5.
54 6344 3E 0435 894 MOVAW (R3)[R4], R4 : Multiply R4 by 2 and add R3.
02 12 0439 895 BNEQ 10$ : If nonzero, quit now.
59 D4 043B 896 CLRL R9 : Reset digit count, since digit
```

BASVAL
2-004

1 8
; Convert text to numeric
MUL10_R9 - multiply FAC by 10 and add

16-SEP-1984 00:01:37 VAX/VMS Macro V04-00
6-SEP-1984 10:39:35 [BASRTL.SRC]BASVAL.MAR;1

Page 24
(16)

05 043D 897
043D 898 10\$: RSB
043E 899

; was not significant.
; Return to caller.

```
043E 901      .SBTTL BAS$VAL_L      ; convert text (integer) to longword
043E 902
043E 903      ;;
043E 904
043E 905      FUNCTIONAL DESCRIPTION:
043E 906
043E 907      BAS$VAL_L converts an ASCII string containing a text
043E 908      representation of a decimal number to internal binary form.
043E 909
043E 910      The text representation converted is:
043E 911      <0 or more blanks>
043E 912      <"+", "-", or nothing>
043E 913      <0 or more ASCII digits from '0' through '9'>
043E 914      <end of string>
043E 915
043E 916      Notes:
043E 917      1. Blanks and tabs are ignored.
043E 918
043E 919      CALLING SEQUENCE:
043E 920
043E 921      status.wlc.v = BAS$VAL_L (in_str.rt.dx )
043E 922
043E 923      INPUT PARAMETERS:
043E 924
043E 925      in_str = 4                      ; Input string by descriptor
043E 926
043E 927      IMPLICIT INPUTS:
043E 928
043E 929      NONE
043E 930
043E 931      OUTPUT PARAMETERS:
043E 932
043E 933      IMPLICIT OUTPUTS:
043E 934
043E 935      value in R0
043E 936
043E 937      COMPLETION CODES:
043E 938
043E 939      SSS NORMAL      - Successful completion
043E 940      BAS$K_ILLNUM    - There was an invalid character in the input
043E 941                      string, the value overflowed the range allowed,
043E 942                      or value_size was invalid. The result "value" is
043E 943                      set to zero, unless value_size is invalid, in which
043E 944                      case "value" is unpredictable.
043E 945
043E 946
043E 947      SIDE EFFECTS:
043E 948
043E 949      NONE
043E 950
043E 951      --
043E 952
043E 953      .ENTRY BAS$VAL_L, REGMASK
043E 954
043E 955      MOVQ    @in_str(AP), R0          ; R0 = width of the input string
043E 956      CLRQ    R4                      ; R1 = address of the input string
043E 957                                     ; R4/R5 = ACC = 0
```

00000004

OFFC

50 04 BC 7D

54 7C


```
56 D4 0446 958 CLR R6 ; clear flags
    0448 959
    0448 960 ;+
    0448 961 ;:-
    0448 962
61 50 20 3B 0448 963 5$: SKPC #^A/ /, R0, (R1) ; skip blanks
    044C 964 ; R0 = #CHAR REMAINING
    044C 965 ; R1 = POINTER_TO_INPUT
    044C 966 ; Z bit is set if R0 = 0
    50 13 044C 967 BEQL DONE_L ; branch to DONE if no non-blank
09 61 91 044E 968 CMPB (R1), #^X09 ; is it a tab?
    06 12 0451 969 BNEQ 7$ ; If not, continue
    51 D6 0453 970 INCL R1 ; Bump pointer
    50 D7 0455 971 DECL R0 ; Decrement counter
    EF 11 0457 972 BRB 5$ ; Look for more.
    20 61 91 0459 973 7$: CMPB (R1), #^A/-/ ; is the current char a '-' sign?
    04 12 045C 974 BNEQ 10$ ; no, branch to 10$
05 56 1F E3 045E 975 BBCS #V_NEGATIVE, R6, DECIMAL_L ; set negative flag and continue
    0462 976
    0462 977
    2B 61 91 0462 978 10$: CMPB (R1), #^A/+/ ; is current char a '+' sign?
    04 12 0465 979 BNEQ DIGIT_LOOP_L ; no, branch to check if it is a digit
    0467 980
    0467 981 ;+
    0467 982 ; skip over '-' or '+' sign
    0467 983 ;:-
    0467 984
    50 D7 0467 985 DECIMAL_L:
    51 D6 0469 986 DECL R0 ; R0 = #CHAR REMAINING
    INCL R1 ; R1 = POINTER_TO_INPUT
```

```
046B 989 :+
046B 990 : Loop to collect digits, treat blanks as zeroes, until the string is exhausted
046B 991 : then branch to DONE
046B 992 :-
046B 993
046B 994 DIGIT_LOOP_L:
50 D7 046B 995      DECL    R0
2F 19 046D 996      BLSS    DONE_L      : R0 = #CHAR REMAINING
046F 997      : branch to DONE if the string is exhausted
046F 998 :+
046F 999 : Get next character, ignoring blanks & tabs.
046F 1000 :-
046F 1001
53 81 9A 046F 1002      MOVZBL (R1)+, R3      : get current char and adjust POINTER_TO_INP
20 53 91 0472 1003      CMPB    R3, #A/ /      : compare char with blank
09 F4 13 0475 1004      BEQL    DIGIT_LOOP_L    : yes, ignore it
53 53 91 0477 1005      CMPB    R3, #X09      : Tab?
EF 13 047A 1006      BEQL    DIGIT_LOOP_L    : Yes, ignore it
00 11 047C 1007      BRB      CHECK_DIGIT_L    : Continue
047E 1008
047E 1009 :+
047E 1010 : Check if current char is a legal digit, accumulate it in ACC if yes and
047E 1011 : then branch to DIGIT_LOOP if no overflow. Otherwise fall into ERROR.
047E 1012 :-
047E 1013
047E 1014 CHECK_DIGIT_L:
53 30 C2 047E 1015      SUBC    #A/0/, R3      : R3 = ASCII(current_char) - ASCII('0')
0E 19 0481 1016      BLSS    ERROR_L      : Error if less than '0'
09 53 91 0483 1017      CMPB    R3, #9      : Is it greater than '9'?
09 14 0486 1018      BGTR    ERROR_L      : If so, error
54 53 54 0A 7A 0488 1019      EMUL    #10, R4, R3, R4      : #10 = radix
048D 1020      : R4 = LP(ACC), only LP(ACC) will be used in
048D 1021      : since R5 (=HP(ACC)) must be zero
048D 1022      : R3 = current digit
048D 1023      : R4/R5 = ACC = ACC * radix + current_digit
55 D5 048D 1024      TSTL    R5      : compare R5 with 0, since a non-zero value
048F 1025      : in HP(ACC) means overflow
DA 13 048F 1026      BEQL    DIGIT_LOOP_L    : if no overflow branch back to get more
0491 1027      : character. Otherwise fall into ERROR
```

```

0491 1029 :+
0491 1030 : ERROR return
0491 1031 :-
0491 1032
0491 1033 ERROR_L:
0491 1034 MOV%9L #BASSK ILLNUM, -(SP)
0495 1035 C- - 5 #1, G^BASS$STOP
049C 1036 BRB EXIT_L ; exit with zero and error
049E 1037
049E 1038 :+
049E 1039 : DONE
049E 1040 :-
049E 1041
049E 1042 DONE_L:
049E 1043 BBC #V_NEGATIVE, R6, 10$ : branch if '-' wasn't seen
04A2 1044 CMPL R4, #^X80000000 : is it 2**31?
04A9 1045 BEQL EXIT_L : yes, already correct!
04AB 1046 TSTL R4 : test for overflow
04AD 1047 BLSS ERROR_L : if already negative, overflow
04AF 1048 MNEGL R4, R4 : answer is -R4
04B2 1049 BRB EXIT_L : Store result
04B4 1050 10$: TSTL R4 : Overflow?
04B6 1051 BLSS ERROR_L : If negative, yes
04B8 1052 EXIT_L:
04B8 1053 MOVL R4, R0 : Move longword result into R0
04BB 1054 RET
04BC 1055

```

```

048C 1057 .SBTTL BASSVAL_P - convert text to packed decimal
048C 1058
048C 1059 :++
048C 1060 : FUNCTIONAL DESCRIPTION:
048C 1061 :
048C 1062 : This routine computes the packed decimal numeric value of an input string
048C 1063 : by calling an RTL conversion routine and returns the value in the
048C 1064 : destination descriptor. If the input string doesn't contain a
048C 1065 : legitimate packed decimal number the routine will signal a
048C 1066 : noncontinuable error.
048C 1067 :
048C 1068 : FORMAL PARAMETERS:
048C 1069 :
048C 1070 : STRING.rt.dx pointer to input string descriptor
048C 1071 : VALUE_DSC.wp.dsd pointer to output packed decimal descriptor
048C 1072 :
048C 1073 : IMPLICIT INPUTS:
048C 1074 :
048C 1075 : NONE
048C 1076 :
048C 1077 : IMPLICIT OUTPUTS:
048C 1078 :
048C 1079 : NONE
048C 1080 :
048C 1081 : ROUTINE VALUE:
048C 1082 :
048C 1083 : NONE
048C 1084 :
048C 1085 : SIDE EFFECTS:
048C 1086 :
048C 1087 : This routine calls the conversion routine and therefore may signal any
048C 1088 : of its errors or have any of its side effects. In particular the
048C 1089 : conversion routine calls STR$ routines and so may allocate or deallocate
048C 1090 : dynamic string space and write lock strings for a short time. It
048C 1091 : may also signal BASSK_ILLNUM if a non-numeric string is input.
048C 1092 :
048C 1093 :--
048C 1094 :
048C 1095 :++
048C 1096 : The following is the Bliss code that this routine was
048C 1097 : generated from.
048C 1098 :
048C 1099 : FMP = .FP;
048C 1100 : DO
048C 1101 : BEGIN ! search back for Basic fram
048C 1102 : FMP = .FMP [BSF$A_SAVED_FP];
048C 1103 : END
048C 1104 : UNTIL (.FMP [BSF$A_HANDLER] EQLA BASS$HANDLER OR
048C 1105 : .FMP EQ 0);
048C 1106 :
048C 1107 : IF BASS$CVT_T_P (STRING [0, 0, 0, 0], ! string to be converted
048C 1108 : VALUE_DSC [0, 0, 0, 0], ! place to put value
048C 1109 : ignore_blanks + ignore_tabs
048C 1110 : + (IF .FMP NEQ 0 AND T.FMP [BSF$W_FCD_FLAGS] AND BSF$M_FCD_RND)
048C 1111 : THEN 0
048C 1112 : ELSE dont_round)) ! flags
048C 1113 : NEQU SS$_NORMAL

```



```
04BC 1114 : THEN
04BC 1115 : BASS$STOP (BASS$ILLNUM); ! input non-numeric, error
04BC 1116 :
04BC 1117 :
04BC 1118 : RETURN
04BC 1119 : END; !End of BASSVAL_P
04BC 1120 :
04BC 1121 :--
04BC 1122 :
OFFC 04BC 1123 : .ENTRY BASSVAL_P, REGMASK
04BE 1124 :
04BE 1125 :+ begin by searching back for a BASIC frame.
04BE 1126 :--
04BE 1127 :
53 52 0C AD D0 04BE 1128 1$: MOVL BSF$A_SAVED_FP(FP), R2 ; get saved frame pointer
00000000'EF 9E 04C2 1129 MOVAB BASS$HANDLER, R3 ;
53 62 D1 04C9 1130 CMPL (R2), R3 ; do we have a BASIC frame?
04 13 04CC 1131 BEQL 2$ ; yes.
52 D5 04CE 1132 TSTL R2 ; no, have we run out of frames?
EC 12 04D0 1133 BNEQ 1$ ; no, keep looking.
04D2 1134 :
04D2 1135 :+
04D2 1136 : we arrive here when we either:
04D2 1137 : - found a BASIC frame, or
04D2 1138 : - we ran out of frames.
04D2 1139 :--
04D2 1140 2$: TSTL R2 ; did we indeed run out of frames?
04D4 1141 BEQL 3$ ; yes.
04E6 A2 09 E1 04D6 1142 BBC #BSF$M_FCD_RND, BSF$W_FCD_FLAGS(R2), 3$ ; no, was "round" bit-flag
04DB 1143 ; set in caller's frame?
04DB 1144 ; if it wasn't, goto 3$
04DB 1145 :
04DB 1146 :+
04DB 1147 : arrive here if:
04DB 1148 : - we found a BASIC frame, and the caller set the "round" bit-flag.
04DB 1149 :--
50 D4 04DB 1150 CLRL R0 ; clear the "don't round" bit
04DD 1151 ; in the flags longword that we
04DD 1152 ; pass to the conversion routine
03 11 04DD 1153 BRB 4$
04DF 1154 :
04DF 1155 :+
04DF 1156 : arrive here if we either:
04DF 1157 : - ran out of frames, or
04DF 1158 : - we found a BASIC frame, but the caller didn't set the "round" flag.
04DF 1159 :--
50 08 D0 04DF 1160 3$: MOVL #dont_round, R0 ; set the "don't round" bit
04E2 1161 ; in the flags longword that
04E2 1162 ; we pass to the conversion
04E2 1163 ; routine
04E2 1164 :
04E2 1165 :+
04E2 1166 : call the routine BASS$CVT_I_P, which actually does the work.
04E2 1167 :
04E2 1168 :
04E2 1169 : note: the "17" in the PUSHAB instruction that follows this is arrived at
04E2 1170 : by adding the symbolic values for "ignore blanks" (1) and "ignore tabs" (16).
```

```

      11 A0 9F 04E2 1171 ;=
      04E2 1172 4$:      PUSHAB 17(R0)      ; take whatever bits we have in
      04E5 1173      ; the flags longword, add the
      04E5 1174      ; "ignore blanks" and "ignore tabs"
      04E5 1175      ; bits to it, and put it
      04E5 1176      ; on the stack
      7E 04 AC 7D 04E5 1177      MOVQ      string(AP), -(SP)      ; put passed string descriptor
      04E9 1178      ; on the stack
00000000'GF 03 FB 04E9 1179      CALLS      #3, G^BASSCVT_T_P      ; call conversion routine
      01 50 D1 04F0 1180      CMPL      R0, #1      ; success?
      0B 13 04F3 1181      BEQL      5$      ; yes, return
      7E 00'8F 9A 04F5 1182      MOVZBL      #BASSK_ILLNUM, -(SP)      ; no, set up to signal error
00000000'GF 01 FB 04F9 1183      CALLS      #1, G^BASS$STOP      ; signal it
      0500 1184
      0500 1185 ;+
      0500 1186 ; all done, return
      0500 1187 ;=
      04 0500 1188 5$:      RET
      0501 1189
      0501 1190
      0501 1191      .END
```

BAS\$VAL
Symbol table

; Convert text to numeric

D 9

16-SEP-1984 00:01:37 VAX/VMS Macro V04-00
6-SEP-1984 10:39:35 [BASRTL.SRC]BASVAL.MAR;1

Page 32
(20)

BAS\$\$SCALE_L_R1	*****	X	00	G_NUM	00000166	R	01
BAS\$\$STOP	*****	X	00	H_EXP	0000019B	R	01
BAS\$CVT_T_P	*****	X	00	H_NUM	0000016A	R	01
BAS\$HANDLER	*****	X	00	INIT_BINEXP	00000180	R	01
BAS\$K_ILNUM	*****	X	00	IN_STR	= 00000004		
BAS\$VAL_D	0000002D	RG	01	K_DTYPE_D	= 00000000		
BAS\$VAL_F	0000001E	RG	01	K_DTYPE_F	= 00000003		
BAS\$VAL_G	0000000F	RG	01	K_DTYPE_G	= 00000001		
BAS\$VAL_H	00000000	RG	01	K_DTYPE_H	= 00000002		
BAS\$VAL_L	0000043E	RG	01	L_2P31_DIV_10	= 0CCCCCCC		
BAS\$VAL_P	000004BC	RG	01	M1	00000431	R	01
BINEXP	= 00000028			M2	0000041A	R	01
BINNUM	= 00000014			M4	000003DA	R	01
BSF\$A_SAVED_FP	= 0000000C			MAKE_VALUE	00000045	R	01
BSF\$M_FCD_RND	= 00000009			MTH\$DINT	*****	X	00
BSF\$W_FCD_FLAGS	= FFFFFFFE			MUL10_R9	000003D1	R	01
CHECK_DIGIT	000000A8	R	01	M_DECEXP	= 10000000		
CHECK_DIGIT_L	0000047E	R	01	M_DEC_POINT	= 40000000		
COMMON	00000054	R	01	M_DIGIT	= 00010000		
CRY	= 00000040			M_NEG_DECEXP	= 20000000		
DECEXP	= 0000000C			NT	000001F9	R	01
DECIMAL_L	00000467	R	01	N2	000001E3	R	01
DECIMAL_POINT	000000E8	R	01	N4	000001C9	R	01
DIGITS	= 00000008			NOT_DIGIT	000000D2	R	01
DIGIT_LOOP	0000009E	R	01	OT\$\$CVT_MUL	*****	X	00
DIGIT_LOOP_L	0000046B	R	01	OT\$\$RET_A_CVT_TAB_R1	*****	X	00
DONE_C	0000049E	R	01	REBASE	0000020D	R	01
DONT_ROUND	= 00000008			REGMASK	= 00000FFC		
DTYPE	= 00000010			REGMASK_H	= 00000FF0		
D_EXP	0000018D	R	01	RGET	000003BB	R	01
D_NUM	00000162	R	01	SCALE	00000151	R	01
ERROR	00000173	R	01	SCALE_VAL	= 00000008		
ERROR_D	000002D3	R	01	STRING	= 00000004		
ERROR_F	000003B7	R	01	TEMP	= 00000000		
ERROR_G	00000318	R	01	UNDERFLOW	0000027C	R	01
ERROR_H	00000378	R	01	V_DEC_POINT	= 0000001E		
ERROR_L	00000491	R	01	V_DIGIT	= 00000010		
EXIT	000003BA	R	01	V_EXT_BITS	= 0000001B		
EXIT_L	000004B8	R	01	V_NEGATIVE	= 0000001F		
EXPON	000000F6	R	01	V_NEG_DECEXP	= 0000001D		
EXPON_DIGITS	000000FE	R	01	ZERO	00000155	R	01
EXP_CHECK	0000012B	R	01	ZERO_RET	00000172	R	01
EXP_COMMON	000001A9	R	01				
EXP_DONE	00000141	R	01				
EXP_LOOP	00000124	R	01				
EXP_NEG	0000011C	R	01				
EXT_BITS	= 00000018						
FLAG	= 00000004						
FLOAT	0000026A	R	01				
FLOAT_D	0000027F	R	01				
FLOAT_F	0000037B	R	01				
FLOAT_G	000002D6	R	01				
FLOAT_H	0000031B	R	01				
FRAME	= 00000050						
F_EXP	000001A2	R	01				
F_NUM	00000170	R	01				
G_EXP	00000194	R	01				

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes															
ABS	00000000 (0.)	00 (0.)	NOPIC	USR	CON	ABS	LCL	NOSHR	NOEXE	NORD	NOWRT	NOVEC	BYTE					
BAS\$CODE	00000501 (1281.)	01 (1.)	PIC	USR	CON	REL	LCL	SHR	EXE	RD	NOWRT	NOVEC	LONG					

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	38	00:00:00.09	00:00:00.63
Command processing	129	00:00:00.42	00:00:01.91
Pass 1	111	00:00:02.76	00:00:06.12
Symbol table sort	0	00:00:00.11	00:00:00.11
Pass 2	203	00:00:02.23	00:00:05.00
Symbol table output	12	00:00:00.09	00:00:00.29
Psect synopsis output	3	00:00:00.02	00:00:00.05
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	498	00:00:05.72	00:00:14.11

The working set limit was 1050 pages.
20176 bytes (40 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 107 non-local and 48 local symbols.
1191 source lines were read in Pass 1, producing 28 object records in Pass 2.
0 pages of virtual memory were used to define 0 macros.

+-----+
! Macro library statistics !
+-----+

Macro library name	Macros defined
_\$255\$DUA28:[SYSLIB]STARLET.MLB;2	0

0 GETS were required to define 0 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS\$:BASVAL/OBJ=OBJ\$:BASVAL MSRC\$:BASVAL/UPDATE=(ENH\$:BASVAL)

0033 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

